The Application and Implementation of Fully Homomorphic Encryption(FHE) Mechanism

Jin Yang, Li Yifan, Yan Yizhou
dept. Department of Computer Science and Engineering
SUSTech

I. Introduction

Homomorphic encryption arise in the rapid development of cloud computing. Four years ago, around 80% of the company with more than 1000 employees adopted cloud computing technology in their production activity and this figure is expected to grow up to 90% in 2024 [1].

Promising as it is, traditional cloud computing has several major draw back. For example, data has to be decrypted before it can be manipulated. Consider user wants to change some figure on the cloud, it must download it from the cloud, decrypt it, perform changes, encrypt it and finally re-upload it. This is inconvenient. On some circumstances, server is not to be trusted. Arbitrary access to the user data is without doubt insecure, as the server administrator could analysis user data to obtain sensitive information or sold the data to third party. But user cannot expect to utilize server's computation without leaking data to the server.

Homomorphic encryption provides a solution to the above dilemma. User first encrypts the data and sent it to the server, who cannot get the plain text. Server then operate on encrypted data and sent it back to the user. When user deciphered it, user will obtain the desired data.

Among various applications, there are privacy database, where the user encrypts its query message and database return the encrypted results. Then after the user decrypts the results it gets the desired data. In this process, the server has no idea what the user is querying. In federated study, each client isomorphically encrypts its model before submitting to the server, thus protecting the confidentiality of data.

II. History Background

In 1978, Rivest, Adleman and Dertouzos proposed a general method for computing on encrypted data. Since proposed, the idea had been a goal in cryptography [2]. It can be seen as the rudiment of Homomorphic Encryption. In 2009 at the international conference STOC, Graig Gentry give the first fully homomorphic encryption solution based on ideal lattices which solved the problem theoretically. Academic and industry circles praised this innovative work, convinced that the Holy Grail exists [3]. In recent years, the research and study of FHE has been more and more popular and important and a lot of results are coming out. For example, Brakerski and Vaikuntanathan implemented FHE with LWE firstly; Gentry et al. realized FHE for the first time by approximating eigenvectors, which is the most classical scheme called Gentry-Sahai-Waters (GSW); Since 2014, iDASH, a top competition in privacy computing is held annually.

III. Definition

What is FHE? It means ciphertexts can be computed arbitrarily. In other words, if we decode the computed ciphertexts, we get the computed plaintext. We write it by cryptography expressions as:

$$f(Enc(m)) = Enc(f(m)) \tag{1}$$

Among (1) m is plaintext, f is a valid function and Enc is the encryption algorithm. $Enc(m)$ is the ciphertext corresponding to m. FHE contains 4 main parts: key generation, encryption, Cipher computing and decryption.

IV. Application Scenarios

FHE allows evaluation of arbitrary functions on encrypted data, so it has a myriad of potential applications [4]. With cloud computing model becoming more and more popular, outsourcing of data storage and computing services has came to a central stage. The problem of data security and privacy protection is getting increasing exposure by the industry and academia [3]. FHE can processing encrypted data without letting them out. For sensitive data such as healthcare information, FHE can be used to enable extended new services by eliminating privacy security issues arising from data sharing. DiNN can combine with FHE. We train model on plaintext, use ciphertext as input and get the encrypted result from neural network. FHE is also used in many scenarios such as machine learning, secure multi-party computing, federated learning, data exchange and sharing.

V. Limitations

It seems that there are several applications which permit FHE as a solution, but there are some limitations about FHE in the real world. We discuss 3 main limitations among them.

First, FHE has difficulties to support multiple users [2]. If there are many users who use the same system which depends on the database for computing want to protect their data from provider, we need many individual databases to solve this problem. It is impossible when their are a mass of users.

Second, their are some problems with complex FHE algorithms. Many FHE algorithms has a lot of computational overhead. This make some FHE applications impractical.

Third, some FHE algorithm can't deal with noise well. It is normal for FHE algorithm to add noise to data during encryption. Noise will grow if we do computation on ciphertexts. Noise will grow faster if we do homomorphic multiplication. When noise grows to a certain extent, decryption algorithm will fail.

VI. Our implementation of FHE application

Many images contain sensitive information and are usually sent to cloud services to encode images for different devices [5]. Cloud is not data safe, so images need encryption.Considering the mentioned conditions, we plan to implement JPEG, an image compression algorithm with FHE on top of Microsoft SEAL library. BFV, BGV and CKKS are the current mainstream FHE algorithms, and we mainly use BFV to implement our project. In our algorithm, a user will first set the security parameters, generate a pair of public key and secret key and encrypts the image with its public key, saving parameters and encrypted data into a file. The executable file doing above work should be built statically so that there is no need for the user to install seal library. Then user sends the encrypted image to the cloud along with the security parameters. Next, server on the cloud will recover the SEAL context from the security parameters and manipulate the encrypted data to perform JPEG compress algorithm and sends it back to the user. Finally, user decrypts the data using its secret key to get the compressed image.

To evaluate our implementation, we consider the correctness and time consumption of the algorithm. Specifically, the decompressed image should be the same as the original image, thus guarantee the correctness. The time consumption should also be within reasonable time.

VII. Application: deploy an encrypted logistic regression service

In order to demonstrate the usage of applying homomorphic encryption in the field of AI to preserve privacy, we now present a way of using logistic regression model to predict over encrypted input. .The dataset we use comes from online database Kaggle (https://www.kaggle.com/code/tanyildizderya/diabetes-prediction-with-logistic-regression), which consists of various health data(age, heart rate, blood pressure, etc) and whether it caused diabates or not. The model it self is trained in plaintext and only the prediction is calculated on encrypted input. This scheme can guarantee that the critical privacy data of the user will not be leaked. The server has the unencrypted model stored in it, which served as the service it provided. The user encrypts its data locally then send it over the network. Without the private key, the server cannot know the actual data. The server computes the output and sent the encrypted result back to user. User decrypts the data with its secret key and get the desired result. The difficulty of adopting homomorphic encryption in AI lies in that we can only perform addition and multiplication over encrypted data. Other common operations like division and comparision is not supported. So its application is quite limited. Non-polynomial functions like sigmoid function can only be approximated by a polynomial by taylor expansion. Logistic regression is applicable because the evaluation can be done by only addition and multiplication. We train the logistic regression model over plaintext in the usual way. Formally, we have a training dataset which consists of $X_1^i, X_2^i, ..., X_n^i$ as the ith data and $Y^i \in \{0, 1\}$ as output. We aim to find a set of parameters $\theta_0, \theta_1, \theta_2, ..., \theta_n$ such that for each $X_1^i, X_2^i, ..., X_n^i$ in the testing dataset, $sigmoid(X^i, \theta)$ can successfully predict Y^i, where $sigmoid(X^i, \theta) = \frac{1}{1+e^{-X^i * \theta}}$. Gradient descend is adopted in the construction of model. Began with some random θs, we improve our model by updating $\theta = \theta + \alpha(y - sigmoid(X, \theta) * X$. To predict, the client can encrypt vector X and send it to the server. The server only need to compute $X * \theta$, a dot product of the input and parameters using homomorphic multiplication and send it back to client. After the client decrypt it, easily compute $sigmoid(X, \theta)$ and predict 1 if the result is greater than 0.5 and 0 otherwise. Experiment shows that the accuracy of encrypted prediction is almost as good as unencrypted prediction at an expense of some extra time.

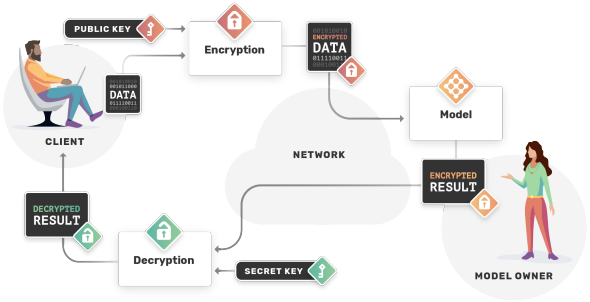

Fig. 1. how the encrypted model work

	accuracy	time consumption(154 tests)
unencrypted	82.47%	0.005s
encrypted	81.17%	2.397s

(test environment: Python3.9
Training data: randomly select 80% of the data from kaggle-diabetes database
Testing data: randomly select 20% of the data from kaggle-diabetes database)

VIII. Division of Work

Yan Yizhou: learn theory knowledge and help other members study relevant knowledge;Help to implement FHE part.

Jin Yang: implement FHE part and help to implement JPEG part; Evaluate the result.

Li Yifan: implement JPEG part and help to implement FHE part; Combine two main part.

IX. Time Line

Week 1-3: make the project team and ensure the topic

Week 4-5: to know about FHE by its history and background

Week 6: study some specific FHE algorithms and use SEAL to write some mini programs

Week 7: prepare for the first project inspection

Week 8-11: study logistic regression and train an encrypted model for it

Week 12:prepare for Second project inspection.

Week 13-15: Improve the image encryption algorithm.

Week 16-17: Debug the whole project to make sure that it work well in most situations.

Week 18: prepare for finally project inspection.

REFERENCES

[1] A. Carey, "On the explanation and implementation of three open-source fully homomorphic encryption libraries," 2020.

[2] F. Armknecht, C. Boyd, C. Carr, K. Gjøsteen, A. Jäschke, C. A. Reuter, and M. Strand, "A guide to fully homomorphic encryption," *Cryptology ePrint Archive*, 2015.

[3] Z. LI, C. MA, and H. ZHOU, "Overview on fully homomorphic encryption," *Journal of Cryptologic Research*, vol. 4, no. 6, pp. 561–578, 2017.

[4] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptology ePrint Archive*, 2012.

[5] W. Fu, R. Lin, and D. Inge, "Fully homomorphic image processing," *arXiv preprint arXiv:1810.03249*, 2018.